Learning from the Rare: Overcoming Class Imbalance in Archaeological Object Detection with Boosting Methods

Argyro Argyrou [1] *, Federico Fasson [2], Emeri Farinetti[2], Apostolos Papakonstantinou[1], Dimitrios Alexakis[3] and Athos Agapiou[1]

[1] Department of Civil Engineering and Geomatics, Faculty of Engineering and Technology, Cyprus University of Technology, Saripolou 2-8, 3036 Achilleos 1 Building, 2nd Floor, Lemesos, Cyprus, P.O. Box. 50329, 7 3603
[2] Department of Humanities, Roma Tre University, via Ostiense 239, 00146 Rome, Italy
[3] Laboratory of Geophysics—Satellite Remote Sensing & Archaeoenvironment (GeoSat ReSeArch Lab), Institute for Mediterranean Studies, Foundation for Research and Technology—Hellas (FORTH), Niki-forou Foka 130 & Melissinou, 74100 Rethymno, Crete, Greece.
* Correspondence: ac.argyrou@edu.cut.ac.cy; Tel.: +35725002542

Highlights

What are the main findings?

- Boosting algorithms (AdaBoost and XGBoost) substantially improved minority-class recall for archaeological potsherd prospection in highly imbalanced datasets, even with only 15% training data—achieving detection-to-ground-truth ratios of 2.5 and 3.2 through deliberate optimization for recall over precision in exploratory contexts.
- Threshold optimization proved essential for archaeological prospection, with optimal thresholds of 0.34 for AdaBoost and 0.11 for XGBoost substantially increasing recall while accepting elevated false positive rates—demonstrating that application-specific tuning is critical when exploratory coverage is prioritized.

What is the implication of the main finding?

- Low-cost UAV surveys provide practical screening tools for flagging high-potential areas requiring field investigation, even with minimal training data—enhancing fieldwalking efficiency by directing efforts toward areas with elevated ceramic probability, particularly valuable for rapid prospection in threatened landscapes.
- Evaluation frameworks for imbalanced archaeological datasets must emphasize recall-based metrics rather than overall accuracy, addressing the "accuracy paradox" where high accuracy masks poor minority-class performance—establishing methodological principles for prospection-oriented remote sensing applicable to rare-artifact detection in cultural heritage contexts requiring field validation.

Abstract

Detecting surface potsherds using low-altitude remote sensing is challenging due to severe class imbalance and limited training data. This study explores boosting algorithms—Adaptive Boosting (AdaBoost) and Extreme Gradient Boosting (XGBoost)—to maximize detection recall for archaeological prospection in the Western Megaris landscape, Greece. Models were trained on only 15% of available data to simulate realistic field conditions. Evaluation emphasized recall-oriented metrics (precision, recall, F1-score, AUC) for the minority class, addressing the accuracy paradox where high overall accuracy masks poor rare-class performance. Threshold optimization enabled AdaBoost and XGBoost to achieve substantially improved recall compared to baseline methods, with detection-to-ground-truth ratios of 2.5 and 3.2 respectively, reflecting deliberate prioritization of recall over precision for exploratory survey purposes. Results

demonstrate that boosting methods with application-specific threshold optimization offer practical screening tools for flagging high-probability areas in archaeological landscape survey, enhancing field survey efficiency in data-constrained environments requiring expert validation.

Keywords: remote sensing; archaeology; accuracy paradox; machine learning; boosting algorithms; surface potsherds; Megara; drone

Acknowledgments: The authors acknowledge the ENSURE project (innovative survey techniques for detection of surface and sub-surface archaeological remains) and CUT internal funding, as well as the Connecting project (SMALL SCALE INFRASTRUCTURES/1222/0062). This paper is part of the PhD dissertation of A. Argyrou, supervised by A. Agapiou, and co-supervised by A. Papakonstantinou and D. Alexakis.

Annex

R script for machine-learning classification and threshold optimisation of raster imagery

```
# --- Libraries ---
library(raster)
library(terra)
library(sp)
library(caret)
library(e1071)
library(randomForest)
library(fastAdaboost)
library(xgboost)
library(pROC)
library(PRROC)
library(ggplot2)
library(openxlsx)
library(reshape2)

# Add this near your safe_round() function
lab2num <- function(x) ifelse(x == "class1", 1, 0)
safe_round <- function(x, digits = 4) {
  if (is.null(x) || is.na(x)) return(NA)
  round(x, digits)
}
# --- Settings ---
set.seed(2025)

# Generic paths (to be adapted by the user)
img_path <- "Input_Image.tif” Input raster (e.g., RGB or multispectral)
csv_path <- "Points.csv” CSV with x, y, Class_ID and other attributes
output_folder <- "Results/” Output folder for results
dir.create(output_folder, showWarnings = FALSE, recursive = TRUE)
```

```
# Raster Output path (same as output folder here)
raster_output_path <- output_folder
dir.create(raster_output_path, showWarnings = FALSE, recursive = TRUE)

# Also use output_path for compatibility with print messages later
output_path <- output_folder

# --- Load Data ---
img <- stack(img_path)
data_csv <- read.csv(csv_path)
coordinates(data_csv) <- ~x + y
crs(data_csv) <- crs(img)
bands <- extract(img, data_csv)

# Rename the first three bands as Band1, Band2, Band3 (generic band names)
if (!is.null(bands) && ncol(bands) >= 3) {
  colnames(bands)[1:3] <- c("Band1", "Band2", "Band3")
}

data_csv@data <- cbind(data_csv@data, bands)
data_csv$Class_ID <- as.factor(data_csv$Class_ID)
data_df <- as.data.frame(data_csv)
data_df[sapply(data_df, is.character)] <- lapply(data_df[sapply(data_df, is.character)],
as.factor)

# --- Stratified split for validation set (20%) ---
set.seed(42)
train_val_idx <- createDataPartition(data_df$Class_ID, p = 0.8, list = FALSE)
data_train_val <- data_df[train_val_idx, ]
data_val <- data_df[-train_val_idx, ]
```

```
# --- From the 80%, 15% for training and 85% for testing ---

set.seed(42)

train_idx <- createDataPartition(data_train_val$Class_ID, p = 0.15, list = FALSE)

train <- data_train_val[train_idx, ]

test <- data_train_val[-train_idx, ]


# --- Fix Factor Levels ---

train$Class_ID <- factor(train$Class_ID, levels = c("0", "1"), labels = c("class0", "class1"))

test$Class_ID <- factor(ifelse(test$Class_ID %in% c(1, "1", "class1"), "class1", "class0"), levels =
c("class0", "class1"))

data_val$Class_ID <- factor(data_val$Class_ID, levels = c("0", "1"), labels = c("class0",
"class1"))


# --- Metrics Function ---

compute_metrics <- function(true, pred, prob) {

  true <- factor(true, levels = c("class0", "class1"))

  pred <- factor(pred, levels = c("class0", "class1"))


  cm <- confusionMatrix(pred, true, positive = "class1")

  TP <- cm$table[2, 2]; TN <- cm$table[1, 1]

  FP <- cm$table[1, 2]; FN <- cm$table[2, 1]


  precision <- if ((TP + FP) == 0) NA else TP / (TP + FP)

  recall <- if ((TP + FN) == 0) NA else TP / (TP + FN)

  specificity <- if ((TN + FP) == 0) NA else TN / (TN + FP)

  sensitivity <- recall

  accuracy <- cm$overall["Accuracy"]

  kappa <- cm$overall["Kappa"]

  mcc_denom <- sqrt((TP+FP)*(TP+FN)*(TN+FP)*(TN+FN))

  mcc <- if (mcc_denom == 0) NA else ((TP*TN)-(FP*FN))/mcc_denom
```

```
  roc_obj <- tryCatch({
    roc(response = true, predictor = prob, levels = c("class0", "class1"), direction = "<")
  }, error = function(e) NA)

  auc_val <- if (is(roc_obj, "roc")) as.numeric(auc(roc_obj)) else NA

  pr_obj <- tryCatch({
    pr.curve(scores.class0 = prob[true == "class1"],
             scores.class1 = prob[true == "class0"],
             curve = TRUE)
  }, error = function(e) NA)

  return(list(
    Accuracy = as.numeric(accuracy),
    Kappa = as.numeric(kappa),
    AUC = auc_val,
    MCC = mcc,
    Precision_class1 = precision,
    Recall_class1 = recall,
    Specificity_class1 = specificity,
    Sensitivity_class1 = sensitivity,
    ROC = roc_obj,
    PR = pr_obj
  ))
}

# --- Model Training ---
ctrl <- trainControl(method = "cv", number = 10, classProbs = TRUE,
                     summaryFunction = twoClassSummary, savePredictions = "final")

models <- list()
```

```
metrics <- list()

# Random Forest
models$rf <- randomForest(Class_ID ~ Band1 + Band2 + Band3, data = train, ntree = 500)
pred_rf <- predict(models$rf, test, type = "response")
prob_rf <- predict(models$rf, test, type = "prob")[, "class1"]
metrics$rf <- compute_metrics(test$Class_ID, pred_rf, prob_rf)
print("Confusion Matrix - RF")
print(confusionMatrix(pred_rf, test$Class_ID, positive = "class1"))

# Maximum Likelihood (Naive Bayes as proxy for MLC)
models$mlc <- naiveBayes(Class_ID ~ Band1 + Band2 + Band3, data = train)
pred_mlc <- predict(models$mlc, test)
prob_mlc <- predict(models$mlc, test, type = "raw")[,"class1"]
metrics$mlc <- compute_metrics(test$Class_ID, pred_mlc, prob_mlc)
print("Confusion Matrix - MLC")
print(confusionMatrix(pred_mlc, test$Class_ID, positive = "class1"))

# SVM
models$svm <- train(Class_ID ~ Band1 + Band2 + Band3, data = train, method = "svmRadial",
                    trControl = ctrl, metric = "ROC",
                    preProc = c("center", "scale"))
pred_svm <- predict(models$svm, test)
prob_svm <- predict(models$svm, test, type = "prob")[,"class1"]
metrics$svm <- compute_metrics(test$Class_ID, pred_svm, prob_svm)
print("Confusion Matrix - SVM")
print(confusionMatrix(pred_svm, test$Class_ID, positive = "class1"))

# AdaBoost
model_ada <- adaboost(Class_ID ~ Band1 + Band2 + Band3, data = train, nIter = 100)
prob_ada <- predict(model_ada, newdata = test, type = "prob")$prob[, 2]
```

```
pred_ada <- factor(ifelse(prob_ada > 0.5, "class1", "class0"), levels = c("class0", "class1"))

cm_ada <- confusionMatrix(pred_ada, test$Class_ID, positive = "class1")

metrics$ada <- compute_metrics(test$Class_ID, pred_ada, prob_ada)

print("Confusion Matrix - ADA")

print(confusionMatrix(pred_ada, test$Class_ID, positive = "class1"))


# XGBoost

dtrain <- xgb.DMatrix(data = as.matrix(train[, c("Band1", "Band2", "Band3")]), label =
as.numeric(train$Class_ID) - 1)

dtest <- xgb.DMatrix(data = as.matrix(test[, c("Band1", "Band2", "Band3")]))

params <- list(objective = "binary:logistic", eval_metric = "logloss")

models$xgb <- xgboost(params = params, data = dtrain, nrounds = 100, verbose = 0)

prob_xgb <- predict(models$xgb, dtest)

pred_class_xgb <- factor(ifelse(prob_xgb >= 0.5, "class1", "class0"), levels = c("class0",
"class1"))

metrics$xgb <- compute_metrics(test$Class_ID, pred_class_xgb, prob_xgb)

print("Confusion Matrix - XGBoost")

print(confusionMatrix(pred_class_xgb, test$Class_ID, positive = "class1"))


# ROC Plot

png(file.path(output_folder, "ROC_Comparison.png"), 800, 600)

plot(metrics$rf$ROC, col = "red", main = "ROC Curves Comparison")

cols <- c("blue", "green", "purple", "orange")

names(cols) <- c("mlc", "svm", "ada", "xgb")

for (mdl in names(cols)) {

  if (!is.null(metrics[[mdl]]$ROC)) lines(metrics[[mdl]]$ROC, col = cols[mdl])

}

legend("bottomright", legend = names(metrics), col = c("red", cols), lwd = 2)

dev.off()


# PR Curve Plot

png(file.path(output_folder, "PR_Comparison.png"), 800, 600)
```

```
plot(metrics$rf$PR$curve[,1:2], type = "l", col = "red", xlab = "Recall", ylab = "Precision", main = "PR Curves Comparison")
for (mdl in names(cols)) {
  if (!is.null(metrics[[mdl]]$PR)) lines(metrics[[mdl]]$PR$curve[,1:2], col = cols[mdl])
}
legend("bottomleft", legend = names(metrics), col = c("red", cols), lwd = 2)
dev.off()

# Save metrics summary to Excel
results_export <- do.call(rbind, lapply(names(metrics), function(m) {
  met <- metrics[[m]]
  data.frame(
    Model = m,
    Accuracy = safe_round(met$Accuracy),
    Kappa = safe_round(met$Kappa),
    AUC = safe_round(met$AUC),
    MCC = safe_round(met$MCC),
    Precision_class1 = safe_round(met$Precision_class1),
    Recall_class1 = safe_round(met$Recall_class1),
    Specificity_class1 = safe_round(met$Specificity_class1),
    Sensitivity_class1 = safe_round(met$Sensitivity_class1)
  )
}))
write.xlsx(results_export, file = file.path(output_folder, "Classification_Metrics_Test.xlsx"), rowNames = FALSE)

# Evaluation on Independent Validation Set (20%)
metrics_val <- list()

# Random Forest
val_pred_rf <- predict(models$rf, data_val)
val_prob_rf <- predict(models$rf, data_val, type = "prob")[, "class1"]
```

```
metrics_val$rf <- compute_metrics(data_val$Class_ID, val_pred_rf, val_prob_rf)
print("Confusion Matrix - RF (Validation)")
print(confusionMatrix(val_pred_rf, data_val$Class_ID, positive = "class1"))

# Naive Bayes
val_pred_mlc <- predict(models$mlc, data_val)
val_prob_mlc <- predict(models$mlc, data_val, type = "raw")[,"class1"]
metrics_val$mlc <- compute_metrics(data_val$Class_ID, val_pred_mlc, val_prob_mlc)
print("Confusion Matrix - MLC (Validation)")
print(confusionMatrix(val_pred_mlc, data_val$Class_ID, positive = "class1"))

# SVM
val_pred_svm <- predict(models$svm, data_val)
val_prob_svm <- predict(models$svm, data_val, type = "prob")[,"class1"]
metrics_val$svm <- compute_metrics(data_val$Class_ID, val_pred_svm, val_prob_svm)
print("Confusion Matrix - SVM (Validation)")
print(confusionMatrix(val_pred_svm, data_val$Class_ID, positive = "class1"))

# AdaBoost
val_prob_ada <- predict(model_ada, newdata = data_val, type = "prob")$prob[,2]
val_pred_ada <- factor(ifelse(val_prob_ada > 0.5, "class1", "class0"), levels = c("class0",
"class1"))
metrics_val$ada <- compute_metrics(data_val$Class_ID, val_pred_ada, val_prob_ada)
print("Confusion Matrix - ADA (Validation)")
print(confusionMatrix(val_pred_ada, data_val$Class_ID, positive = "class1"))

# XGBoost
val_prob_xgb <- predict(models$xgb, newdata = xgb.DMatrix(as.matrix(data_val[, c("Band1",
"Band2", "Band3")])))
val_pred_xgb <- factor(ifelse(val_prob_xgb > 0.5, "class1", "class0"), levels = c("class0",
"class1"))
metrics_val$xgb <- compute_metrics(data_val$Class_ID, val_pred_xgb, val_prob_xgb)
```

```
print("Confusion Matrix - XGBoost (Validation)")
print(confusionMatrix(val_pred_xgb, data_val$Class_ID, positive = "class1"))

# Construct validation metrics table
metrics_val_table <- do.call(rbind, lapply(names(metrics_val), function(m) {
  met <- metrics_val[[m]]
  data.frame(
    Model = m,
    Accuracy = safe_round(met$Accuracy),
    Kappa = safe_round(met$Kappa),
    AUC = safe_round(met$AUC),
    MCC = safe_round(met$MCC),
    Precision_class1 = safe_round(met$Precision_class1),
    Recall_class1 = safe_round(met$Recall_class1),
    Specificity_class1 = safe_round(met$Specificity_class1),
    Sensitivity_class1 = safe_round(met$Sensitivity_class1)
  )
}))
write.xlsx(metrics_val_table, file = file.path(output_folder,
"Classification_Validation_Final.xlsx"), rowNames = FALSE)

# Save images
cat("\nAll metrics (including validation set) and images saved to:", output_path, "\n")

# Load raster (first three bands as Band1, Band2, Band3)
r_rgb <- rast(img_path)[[1:3]]
names(r_rgb) <- c("Band1", "Band2", "Band3")  # Same feature names as in the training set

# Set paths for raster predictions
output_preds <- list(
  rf  = file.path(output_path, "classified_rf.tif"),
```

```
  mlc = file.path(output_path, "classified_mlc.tif"),
  svm = file.path(output_path, "classified_svm.tif"),
  ada = file.path(output_path, "classified_ada.tif"),
  xgb = file.path(output_path, "classified_xgb.tif")
)

# Predict RF (full-scene)
terra::predict(
  r_rgb,
  models$rf,
  filename = output_preds$rf,
  datatype = "INT1U",
  overwrite = TRUE,
  progress = 10
)

# Predict MLC (Naive Bayes) full-scene
terra::predict(
  r_rgb,
  models$mlc,
  filename = output_preds$mlc,
  datatype = "INT1U",
  overwrite = TRUE,
  progress = 10
)

# Predict SVM full-scene
r_pred_svm <- app(
  r_rgb,
  fun = function(d) {
    p <- predict(models$svm, newdata = as.data.frame(d), type = "prob")[, "class1"]
```

```
    lab2num(ifelse(p >= 0.5, "class1", "class0"))
  }
)

writeRaster(r_pred_svm,
            filename = output_preds$svm,
            datatype = "INT1U",
            overwrite = TRUE)

# Predict AdaBoost with threshold = 0.5 (initial)
r_pred_ada <- app(r_rgb, fun = function(d) {
  p <- predict(model_ada, newdata = as.data.frame(d), type = "prob")$prob[, 2]
  lab2num(ifelse(p >= 0.5, "class1", "class0"))
})

writeRaster(r_pred_ada,
            filename = output_preds$ada,
            datatype = "INT1U",
            overwrite = TRUE)

# Predict XGBoost with threshold = 0.5 (initial)
r_pred_xgb <- app(r_rgb, fun = function(d) {
  m <- as.matrix(d)
  p <- predict(models$xgb, newdata = xgb.DMatrix(m))
  lab2num(ifelse(p >= 0.5, "class1", "class0"))
})

writeRaster(r_pred_xgb, filename = output_preds$xgb, datatype = "INT1U", overwrite = TRUE)

message("All initial full-scene classifications (RF, MLC, SVM, AdaBoost, XGBoost) completed
and saved in folder: ", output_path)
```

```
val_true <- factor(data_val$Class_ID, levels = c("class0", "class1"))


# --- Threshold tuning ---
evaluate_thresholds <- function(probs, true_labels, model_name, thresholds = seq(0.05, 0.95, by = 0.01)) {
  results <- data.frame(
    Threshold = thresholds,
    TP = NA, FP = NA, FN = NA, TN = NA,
    Precision = NA, Recall = NA, F1 = NA, Accuracy = NA,
    Specificity = NA, Sensitivity = NA, MCC = NA
  )

  for (i in seq_along(thresholds)) {
    thr <- thresholds[i]
    preds <- factor(ifelse(probs >= thr, "class1", "class0"), levels = c("class0", "class1"))
    cm <- confusionMatrix(preds, true_labels, positive = "class1")

    TP <- cm$table[2, 2]; FP <- cm$table[1, 2]
    FN <- cm$table[2, 1]; TN <- cm$table[1, 1]

    precision <- if ((TP + FP) == 0) NA else TP / (TP + FP)
    recall    <- if ((TP + FN) == 0) NA else TP / (TP + FN)
    f1        <- if (is.na(precision) || is.na(recall) || (precision + recall) == 0) NA else 2 * precision * recall / (precision + recall)
    accuracy  <- cm$overall["Accuracy"]
    specificity <- if ((TN + FP) == 0) NA else TN / (TN + FP)
    sensitivity <- recall

    mcc_denom <- sqrt((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN))
    mcc <- if (mcc_denom == 0) NA else ((TP * TN) - (FP * FN)) / mcc_denom
```

```
    results[i, 2:12] <- c(TP, FP, FN, TN, precision, recall, f1, accuracy, specificity, sensitivity, mcc)
  }

  best_f1_row <- results[which.max(results$F1), ]
  best_mcc_row <- results[which.max(results$MCC), ]

  cat(paste0("Best threshold for ", model_name, ":\n"))
  cat(paste0("- Max F1  = ", best_f1_row$Threshold, "\n"))
  cat(paste0("- Max MCC = ", best_mcc_row$Threshold, "\n"))

  list(
    metrics = results,
    best_threshold_f1 = best_f1_row$Threshold,
    best_threshold_mcc = best_mcc_row$Threshold
  )
}

# Apply Threshold tuning (AdaBoost & XGBoost) on validation probabilities
val_true <- data_val$Class_ID

results_ada <- evaluate_thresholds(val_prob_ada, val_true, "AdaBoost")
results_xgb <- evaluate_thresholds(val_prob_xgb, val_true, "XGBoost")

# Results to Excel ---
write.xlsx(
  list(
    AdaBoost = results_ada$metrics,
    XGBoost = results_xgb$metrics
  ),
  file = file.path(output_folder, "Threshold_Tuning_Metrics.xlsx"),
  rowNames = FALSE
```

```
)

# Create plots F1 & MCC vs Threshold
plot_threshold_curves <- function(df, model_name) {
  df_long <- reshape2::melt(df[, c("Threshold", "F1", "MCC")], id.vars = "Threshold")

  ggplot(df_long, aes(x = Threshold, y = value, color = variable)) +
    geom_line() + geom_point() +
    labs(title = paste("F1 & MCC vs Threshold -", model_name),
         y = "Score", color = "Metric") +
    theme_minimal()
}

ggsave(file.path(output_folder, "F1_MCC_vs_Threshold_AdaBoost.png"),
       plot_threshold_curves(results_ada$metrics, "AdaBoost"), width = 6, height = 4)

ggsave(file.path(output_folder, "F1_MCC_vs_Threshold_XGBoost.png"),
       plot_threshold_curves(results_xgb$metrics, "XGBoost"), width = 6, height = 4)

# Threshold AdaBoost (using fixed example best threshold)
best_thr_ada <- 0.34
model_ada <- adaboost(Class_ID ~ Band1 + Band2 + Band3, data = train, nIter = 100)
prob_ada <- predict(model_ada, newdata = test, type = "prob")$prob[, 2]
pred_ada <- factor(ifelse(prob_ada >= best_thr_ada, "class1", "class0"), levels = c("class0",
"class1"))
cm_ada <- confusionMatrix(pred_ada, test$Class_ID, positive = "class1")
metrics$ada <- compute_metrics(test$Class_ID, pred_ada, prob_ada)
print("Confusion Matrix - ADA (Best threshold)")
print(cm_ada)

# Threshold XGBoost
```

```
best_thr_xgb <- 0.11

models <- list()  # Create list if not already existing


dtrain <- xgb.DMatrix(data = as.matrix(train[, c("Band1", "Band2", "Band3")]), label = as.numeric(train$Class_ID) - 1)

dtest <- xgb.DMatrix(data = as.matrix(test[, c("Band1", "Band2", "Band3")]))

params <- list(objective = "binary:logistic", eval_metric = "logloss")

models$xgb <- xgboost(params = params, data = dtrain, nrounds = 100, verbose = 0)


prob_xgb <- predict(models$xgb, dtest)

pred_class_xgb <- factor(ifelse(prob_xgb >= best_thr_xgb, "class1", "class0"), levels = c("class0", "class1"))

metrics$xgb <- compute_metrics(test$Class_ID, pred_class_xgb, prob_xgb)

print("Confusion Matrix - XGBoost (Best threshold)")

print(confusionMatrix(pred_class_xgb, test$Class_ID, positive = "class1"))


# --- Validation Set (AdaBoost with best threshold) ---

val_prob_adaT <- predict(model_ada, newdata = data_val, type = "prob")$prob[,2]

val_pred_adaT <- factor(ifelse(val_prob_adaT >= best_thr_ada, "class1", "class0"), levels = c("class0", "class1"))

metrics_val$ada <- compute_metrics(data_val$Class_ID, val_pred_adaT, val_prob_adaT)

print("Confusion Matrix - ADA (Validation, best threshold)")

print(confusionMatrix(val_pred_adaT, data_val$Class_ID, positive = "class1"))


# --- Validation Set (XGBoost with best threshold) ---

val_prob_xgbT <- predict(models$xgb, newdata = xgb.DMatrix(as.matrix(data_val[, c("Band1", "Band2", "Band3")])))

val_pred_xgbT <- factor(ifelse(val_prob_xgbT >= best_thr_xgb, "class1", "class0"), levels = c("class0", "class1"))

metrics_val$xgb <- compute_metrics(data_val$Class_ID, val_pred_xgbT, val_prob_xgbT)

print("Confusion Matrix - XGBoost (Validation, best threshold)")

print(confusionMatrix(val_pred_xgbT, data_val$Class_ID, positive = "class1"))
```

```
# --- Example: AdaBoost full-scene classification with chosen threshold (separate generic block)
---

# Required packages
library(terra)

# Inputs (generic)
# img_path already defined above
# output_path already defined above
best_model_ada <- model_ada  # trained AdaBoost model with chosen threshold
# best_thr_ada already defined above

# Load raster as Band1, Band2, Band3
r_rgb <- rast(img_path)[[1:3]]
names(r_rgb) <- c("Band1", "Band2", "Band3")

# Predict and classify raster using AdaBoost + threshold
r_pred_ada_thr <- app(r_rgb, fun = function(d) {
  p <- predict(best_model_ada, newdata = as.data.frame(d), type = "prob")$prob[, 2]
  lab2num(ifelse(p >= best_thr_ada, "class1", "class0"))
})

# Save output GeoTIFF (0 = class0, 1 = class1)
output_tif_ada_thr <- file.path(output_path, "classified_ada_threshold.tif")

writeRaster(r_pred_ada_thr,
            filename = output_tif_ada_thr,
            datatype = "INT1U",  # 1-byte unsigned integer (values 0–255)
            overwrite = TRUE,
            wopt = list(gdal = "COMPRESS=LZW"))

cat("GeoTIFF (AdaBoost with threshold) saved to:", output_tif_ada_thr, "\n")
```

```
# --- Example: XGBoost full-scene classification with chosen threshold (stars-based block) ---
library(stars)
library(xgboost)

# Inputs (generic; img_path and output_path already defined)
best_thr_xgb <- best_thr_xgb
best_model_xgb <- models$xgb

# Load raster as stars object
r_rgb_stars <- read_stars(img_path, proxy = FALSE)

# Keep only first 3 bands (assuming Band1, Band2, Band3)
r_rgb_stars <- r_rgb_stars[,,,1:3, drop=TRUE]

# Flatten raster to matrix (rows = pixels, columns = bands)
mat <- matrix(r_rgb_stars[[1]], ncol = 3, byrow = FALSE)

# Remove NA rows (incomplete cases)
valid_idx <- complete.cases(mat)

# Prepare output vector
pred_vec <- rep(NA_integer_, nrow(mat))

# Predict for valid pixels
dmat <- xgb.DMatrix(data = mat[valid_idx, , drop=FALSE])
p <- predict(best_model_xgb, newdata = dmat)

# Apply threshold
pred_vec[valid_idx] <- as.integer(p >= best_thr_xgb)
```

```
# Convert prediction back to stars raster
dim(pred_vec) <- dim(r_rgb_stars)[1:2]
r_pred_xgb_thr <- st_as_stars(pred_vec)

# Copy geospatial metadata
st_crs(r_pred_xgb_thr) <- st_crs(r_rgb_stars)
st_dimensions(r_pred_xgb_thr) <- st_dimensions(r_rgb_stars)[1:2]

# Write to GeoTIFF
output_file_xgb_thr <- file.path(output_path, "classified_xgb_threshold.tif")
write_stars(r_pred_xgb_thr, output_file_xgb_thr)

cat("XGBoost thresholded classification saved:", output_file_xgb_thr, "\n")
```